Relazione per “Programmazione ad Oggetti”

Quaranta Federico,
Pezzolesi Luca,
Valente Domenico

24 febbraio 2026

Indice

Capitolo 1

Analisi

1.1 Descrizione e requisiti

Il videogioco "C.F.U." si pone come obiettivo il raggiungimento dell'ambita laurea mancante del player; suddetta laurea sarà ottenibile mediante l'esplorazione del campus di Cesena nel quale si è rimasti intrappolati. Muovendosi all'interno del campus saranno reperibili oggetti che permetteranno interazioni con l'ambiente (ad es. chiavi per aprire porte).
Per raggiungere la laurea non sarà necessario risolvere tutti gli enigmi, quella di risolverli tutti rimarrà una scelta personale.

Requisiti funzionali

- il player deve potersi spostare nell'ambiente liberamente e interagirci in maniera funzionale(es: impossibilità di attraversamento ostacoli, salvataggio oggetti ecc..);
- deve essere possibile rispondere agli enigmi e poterli interrompere;
- deve essere presente un inventario dove verranno salvati oggetti utilizzabili per interagire con l'ambiente;

Requisiti non funzionali

- il software deve essere multipiattaforma, garantendo l'esecuzione e la giocabilità su diversi sistemi operativi;
- il programma deve poter essere eseguito anche su hardware datato (con un'anzianità massima di 10 anni), continuandone a garantire la giocabilità.

1.2 Modello del Dominio

Il player dovrà accedere ad un insieme di stanze (ambiente). Ogni stanza contiene enigmi e/o oggetti, con i quali il player potrà interagire, utili allo sblocco di aree precedentemente inaccessibili, avanzando nel percorso per poi trovare l'uscita. In ogni stanza saranno presenti diversi enigmi, risolvibili tramite selezione di una delle alternative proposte come risposta alla domanda dell'enigma. Non tutti gli enigmi saranno necessari per il proseguimento del percorso, ma ognuno contribuirà al risultato finale. Solo un enigma per ogni stanza conterrà una "chiave" utile per lo sblocco della stanza successiva. Una volta ottenuta questa chiave, la porta si sbloccherà automaticamente. Sarà possibile tenere traccia del percorso fatto tramite l'inventario, dove saranno presenti le chiavi ottenute.

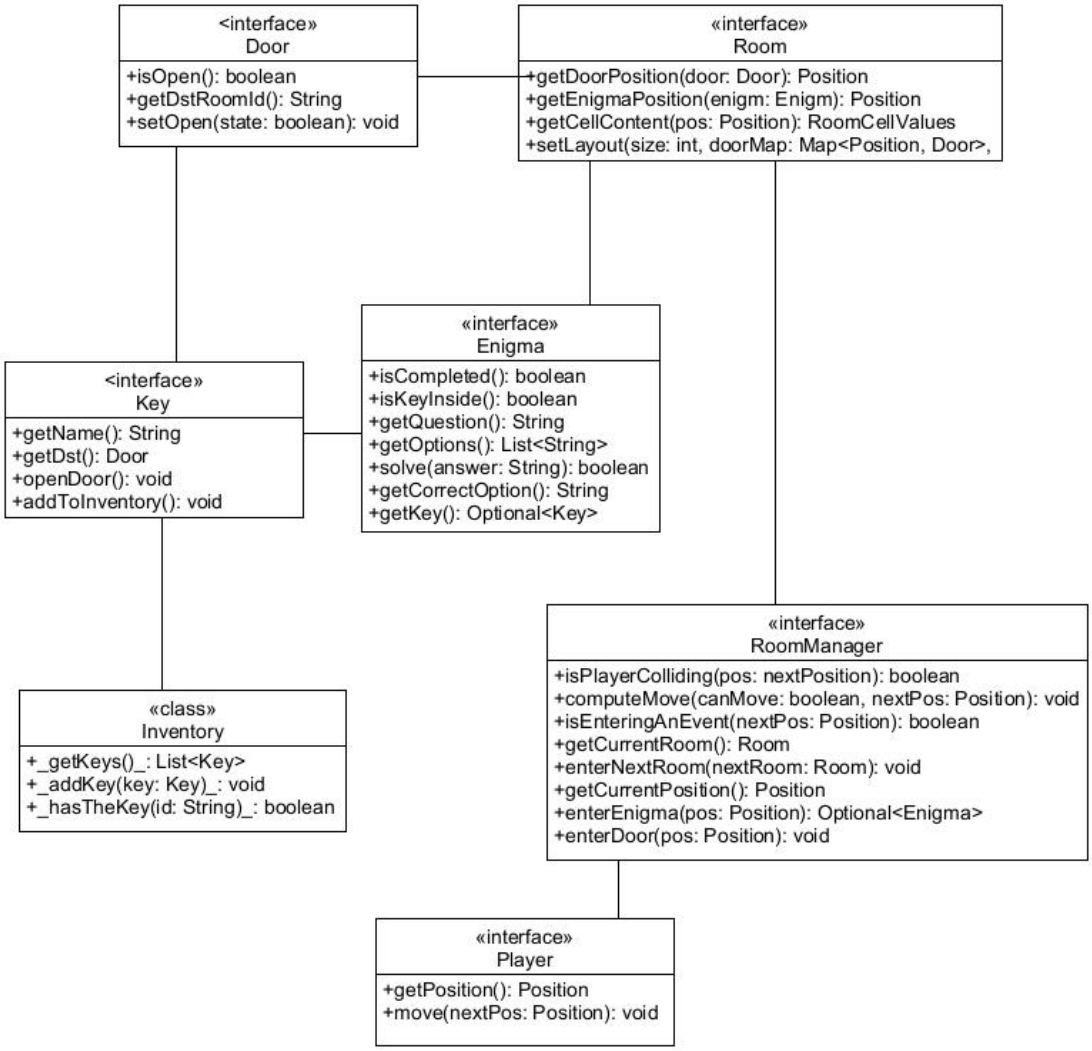

Figura 1.1: Schema UML dell'analisi del dominio, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

Il videogioco segue il pattern architetturale MVC, per separare al meglio la parte logica, di gestione dell'input e dell'interfaccia grafica. La parte del model comprende tutte le componenti che rappresentano la logica del gioco e garantisce l'applicazione delle regole del gioco in modo corretto. Il Controller agisce da intermediario, coordinando la cattura dell'input con la risposta da mettere in output. La View si occupa esclusivamente dei meccanismi di cattura dell'input e visualizzazione dell'output, per cui non dovrà dipendere né dal controller né dal model.

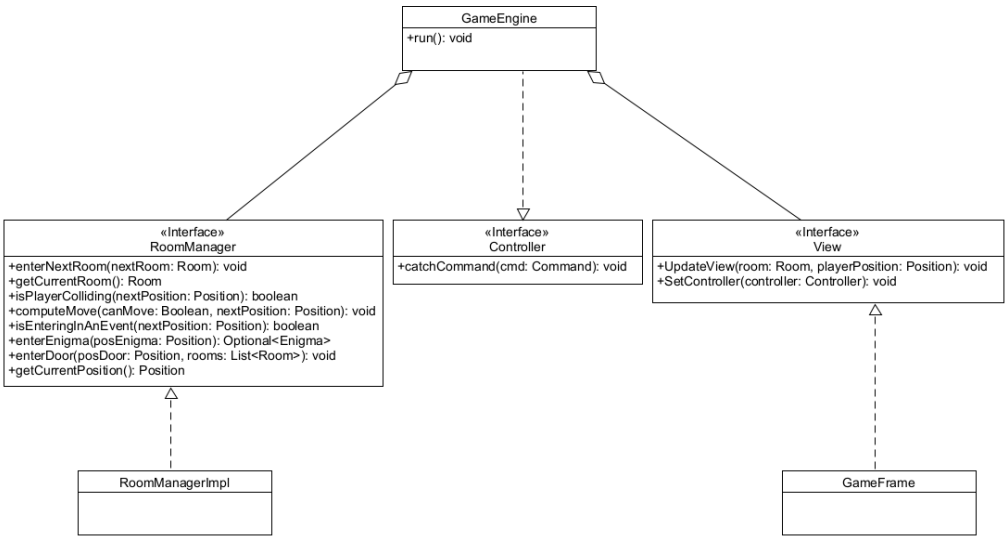

Figura 2.1: Controller è il controller generale del videogioco, RoomManager rappresenta la parte puramente logica, GameFrame la parte grafica

2.2 Design dettagliato

2.2.1 Quaranta Federico

- Implemntazione della logica di gestioone delle collisioni tra player e oggetti nella mappa;
- Gestione delle conseguenze derivanti dall'interazione tra player ed eventi (es: cambio stanza);
- Creazione di un sistema di inventario.

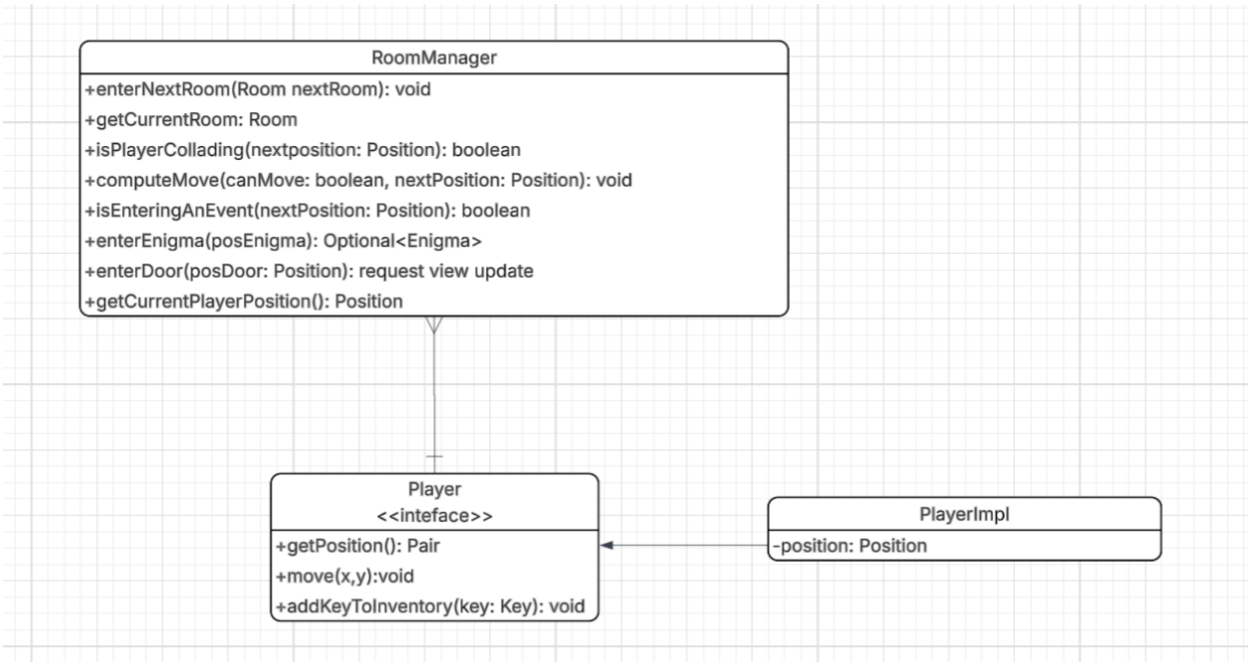

Figura 2.2:

Problema: deve essere possibile che il RoomManager gestisca le collisioni che il player ha con gli elementi presenti nella mappa, governando così le conseguenze che ogni interazione genera.
Soluzione: il RoomManager implementa metodi di controllo sul tipo di collisione che sta avvenendo, comunicando al player come comportarsi. Tramite il player, il Room manager riesce anche a comunicare con l'Inventario, permettendone la gestione di esso.
Alternative:
-approccio monolitico: una classe player che gestisce variabili, collisioni e stampe a video
-pro: rapida implementazione iniziale
-contro: difficilmente manutenibile
-pattern observer: il model comunica direttamente alla view il cambiamento dei dati:
- pro: massimo disaccoppiamento (classi indipendenti)
-contro: complessità eccessiva

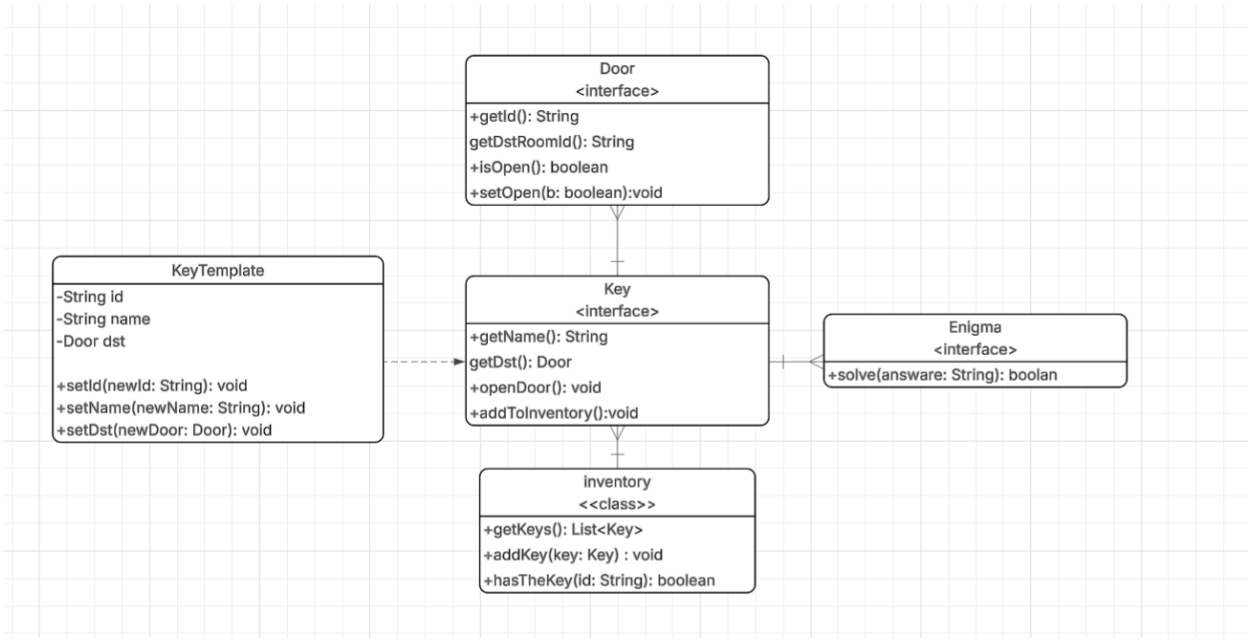

Figura 2.3:

Problema: Aggiornare lo stato dell'inventario nel momento in cui viene trovata una chiave e, simultaneamente, sbloccare la porta corrispondente.

Soluzione: All'interno della classe Enigma è presente un metodo solve(). Quando l'utente fornisce la risposta corretta, solve() attiva due processi:
Richiama il metodo openDoor() della chiave associata (se presente).
Esegue il metodo addToInventory().
Nello specifico, openDoor() agisce sulla classe Door tramite il metodo setOpen(true), mentre addToInventory() interagisce con la classe statica Inventory chiamando addKey(this). I metodi setter presenti nella classe Key sono riservati esclusivamente alla gestione della persistenza tramite YAML.

2.2.2 Valente Domenico

- Implementazione del sistema di gestione modulare dell'ambiente di gioco;
- Sistema di realizzazione dei tre tipi di enigmi e creazione di un modello standard per gli enigmi;
- Creazione del game engine, in particolare gestione del tempo all'interno del gioco.

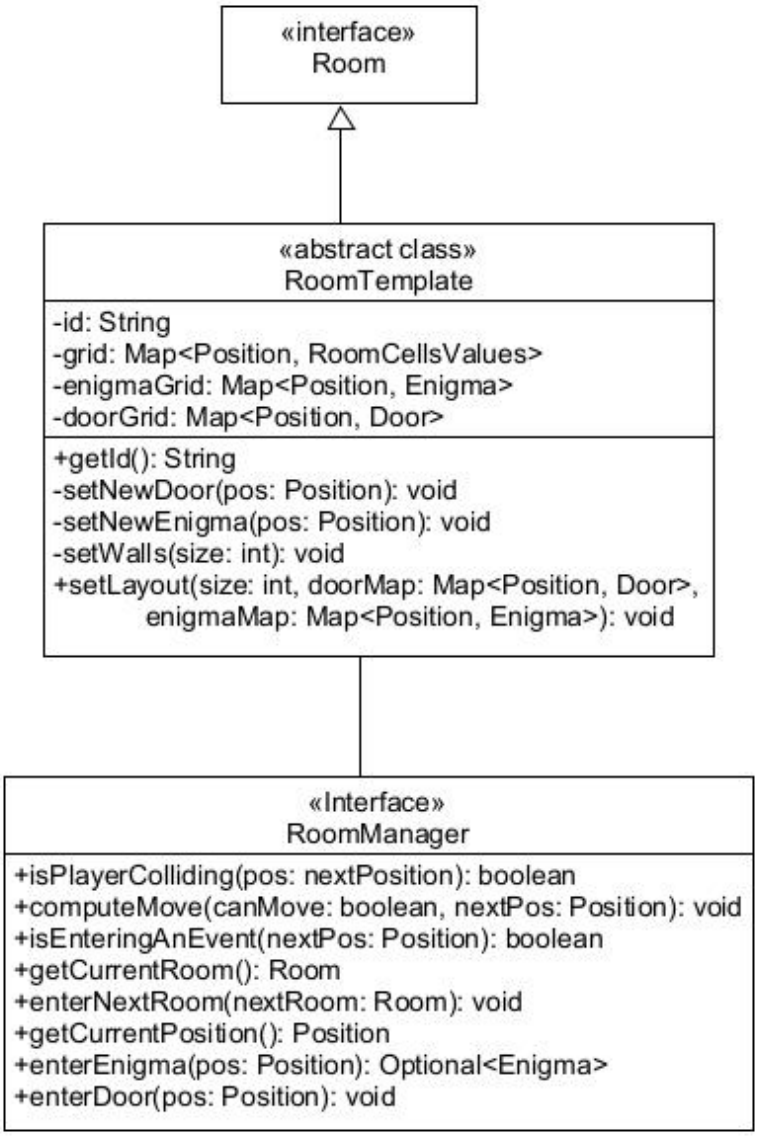

Figura 2.4: Rappresentazione UMLdi un pattern basato sul "Template Method" per la gestione dell'ambiente di gioco. Problema: in ogni stanza (Room), la disposizione degli oggetti di gioco dovrà sempre essere diversa. Soluzione: creazione di un Template, su cui costruire velocemente nuove stanze e cambiarne il layout. Questa soluzione permette di poter implementare in modo semplice più tipi di stanze, garantendo un discreto riuso del codice.

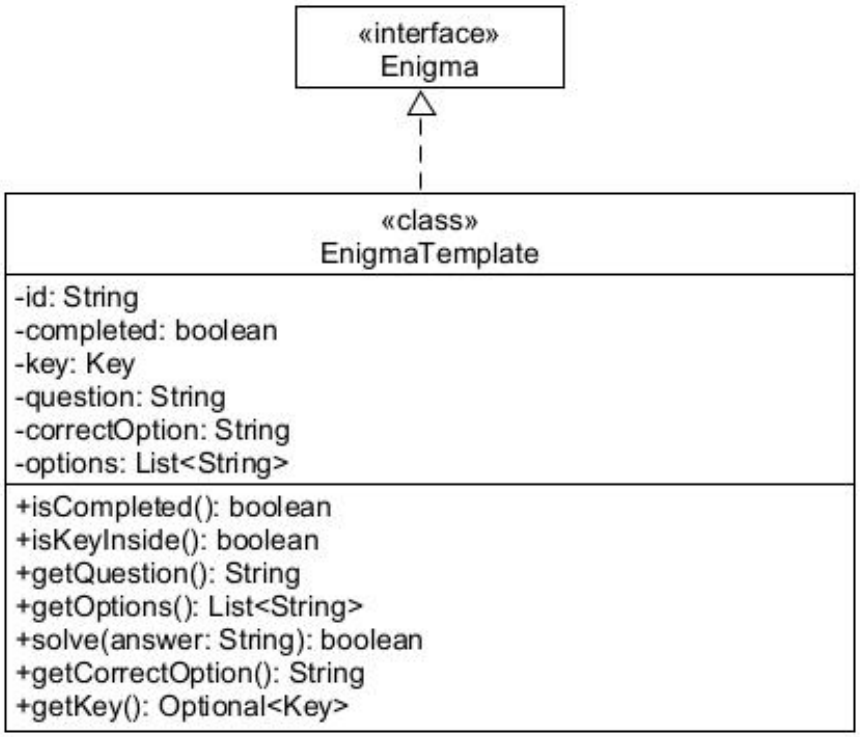

Figura 2.5: Rappresentazione UML di un pattern basato sul "Template Method" per il sistema di creazione di nuovi enigmi. Problema: come le stanze dell'ambiente di gioco, ogni enigma dovrà avere domande e soluzioni diverse, e deve essere intuitivo e semplice creare enigmi di tipi diversi. Soluzione: creazione di un template comune a tutti gli enigmi che gestisce le operazioni di base: porre le domande e verificare le risposte. In questo modo, viene garantito il riuso e la pulizia del codice.

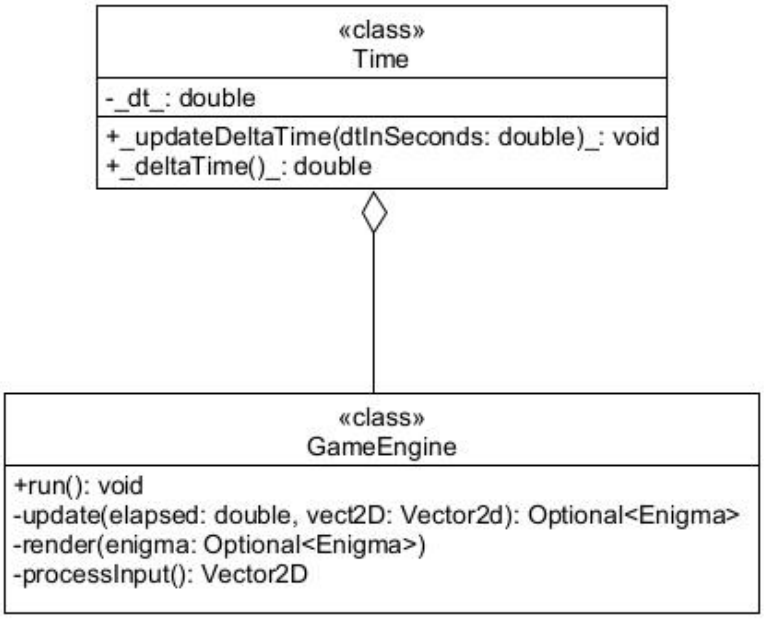

Figura 2.6: Rappresentazione UML del game engine e gestione dei periodi tra frame

2.2.3 Pezzolesi Luca

- implementazione dei commandi;
- implementazione di un sistema di gestione del salvataggio del gioco;
- Creazione della view(GUI).

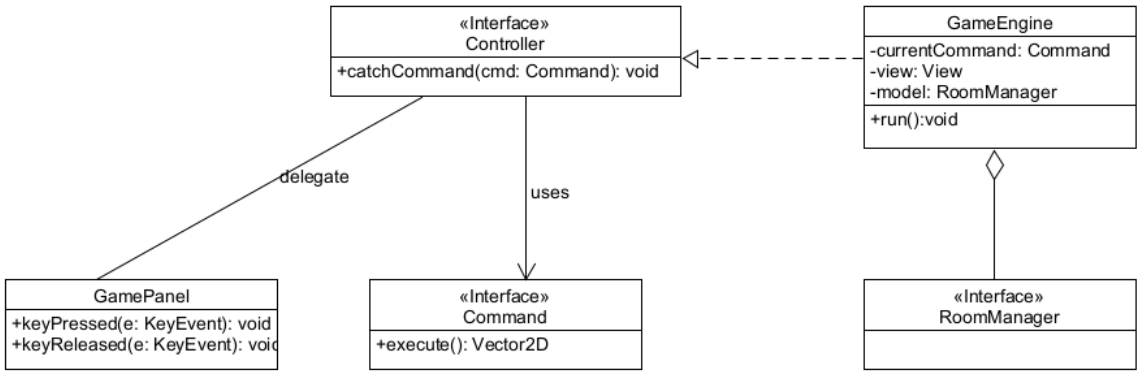

Figura 2.7: Problema: Nel sistema è necessario gestire diverse azioni (comandi) che il controller deve poter eseguire in risposta all'input dell'utente. Nel nostro caso, l'interazione avviene tramite tastiera per il movimento del giocatore, che deve attivarsi alla pressione di un tasto (le quattro frecce direzionali) e interrompersi al rilascio dello stesso.

Soluzione: La soluzione adottata sfrutta la gestione degli eventi: la View, basata su Swing, intercetta la pressione e il rilascio dei tasti e delega l'azione al Controller. Quest'ultimo traduce l'evento in un oggetto che implementa l'interfaccia Command e lo registra tramite il metodo catchCommand(). Il comando viene successivamente eseguito tramite il metodo execute() all'interno del metodo run() del GameEngine, che agisce sul GameModel modificandone lo stato in modo coerente con l'input ricevuto.

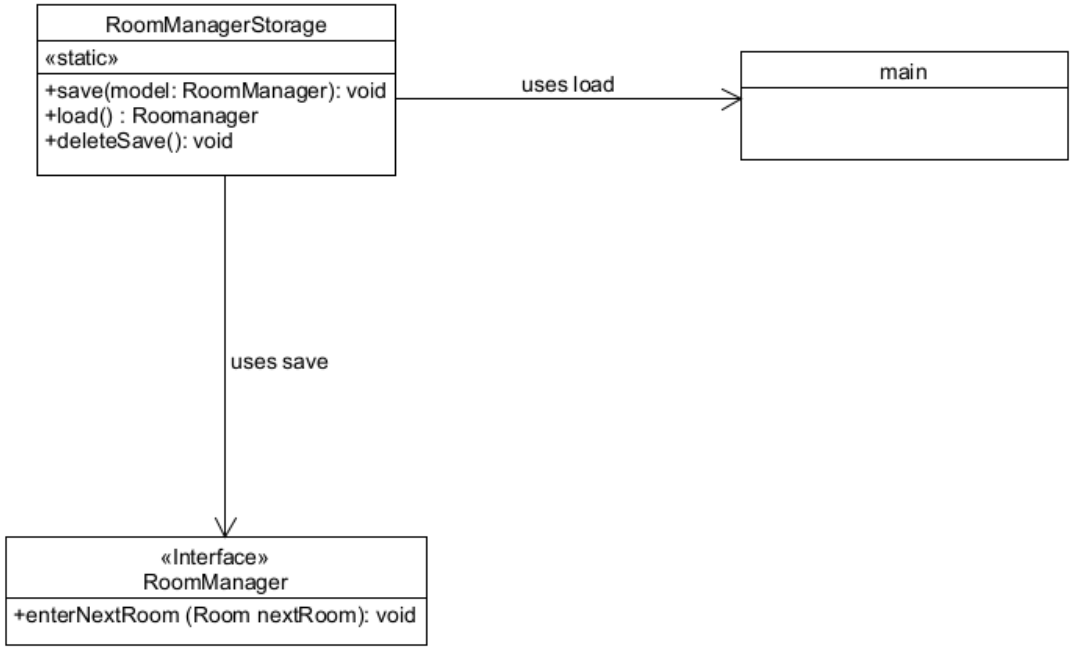

Figura 2.8: Problema: Nel sistema è necessario prevedere un meccanismo di salvataggio e caricamento dello stato del modello (RoomManager).

Soluzione: La soluzione adottata consiste nell'introduzione di una classe statica RoomManagerStorage, responsabile esclusivamente della persistenza dello stato del RoomManager. Essa fornisce un metodo save(), utilizzato dal metodo enterNextDoor() di RoomManager per salvare automaticamente il gioco a ogni cambio di stanza, e un metodo load(), invocato dal main all'avvio dell'applicazione per ripristinare l'ultimo stato salvato, qualora disponibile.

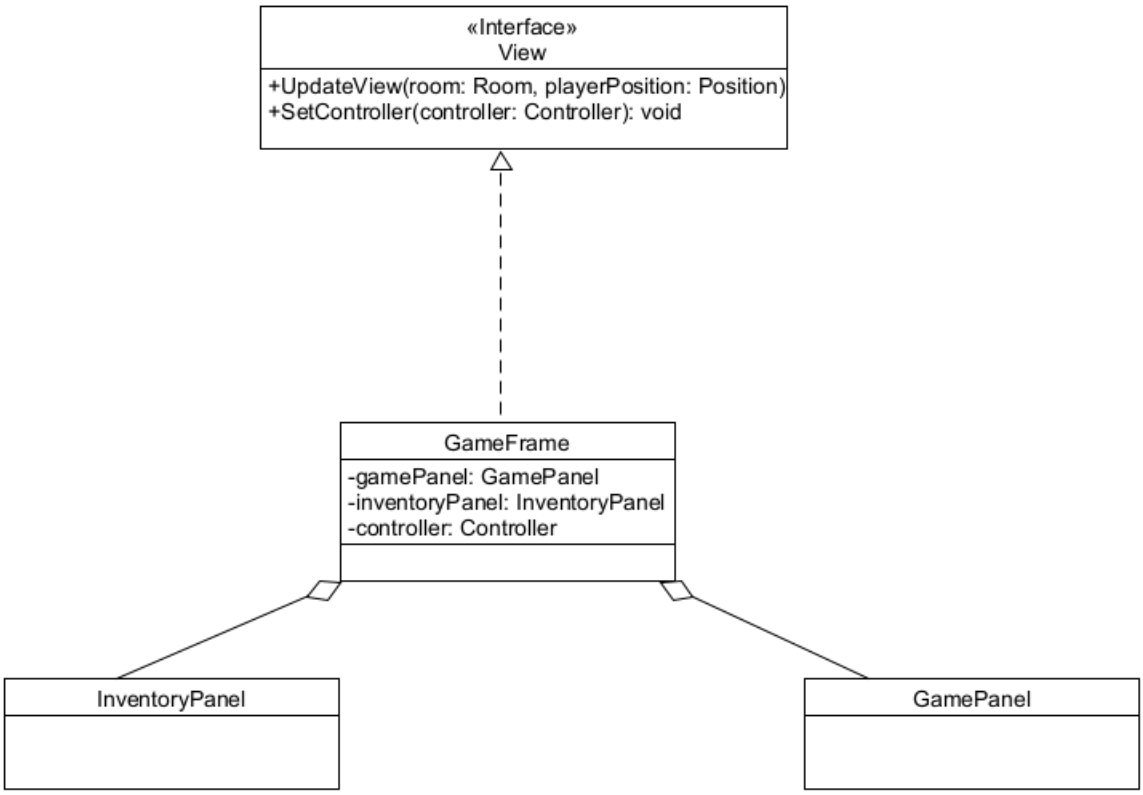

Figura 2.9: Problema: Nel sistema sorge la necessità di rendere la logica di visualizzazione intercambiabile e indipendente dall'implementazione tecnologica specifica (es. Java Swing).

Soluzione: Per massimizzare la modularità e favorire l'indipendenza tecnologica, è stata introdotta l'interfaccia View, che definisce un contratto comune basato sui metodi UpdateView e SetController. L'implementazione concreta è affidata alla classe GameFrame, che realizza l'interfaccia fungendo da contenitore principale. La struttura interna segue un principio di composizione, GamePanel: gestisce la resa grafica del gioco e l'interazione con l'utente. Invece InventoryPanel: si occupa specificamente della visualizzazione dell'inventario. Questa architettura permette di sostituire l'intero blocco grafico (ad esempio passando da Swing a JavaFX) semplicemente creando una nuova classe che implementi View, senza dover modificare il controller.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Abbiamo deciso di usare JUinit per la parte di testing automatizzato.

3.1.1 Sistema di inventario

Test di funzionamento inventario: testing della memorizzazione degli oggetti al suo interno.

3.1.2 Componenti dell'ambiente di gioco

Testing della corretta generazione delle Rooms e del corretto lancio delle eccezioni

3.1.3 Player

Testing su inizializzazione del Player, aggiornamento della sua posizione eaggiornamento dei punti.

3.1.4 Memorizzazione e lettura dei dati su file

Testing delle funzioni di scrittura e lettura dei salvataggi su file.

3.2 Note di sviluppo

3.2.1 Valente Domenico

Lambda expressions

lambda expression senza argomenti per testare il corretto lancio di eccezioni:
https://github.com/2menc/OOP25-C_F_U_/blob/5ee38ae834dff60ca6f23568f8a3136024326
src/test/java/it/unibo/mapComponents/TestRoom.java#L53

streams

stream per riconvertire i dati salvati in oggetti "Enigma": https://github.
com/2menc/OOP25-C_F_U_/blob/5ee38ae834dff60ca6f23568f8a31360243269c7/
src/main/java/it/unibo/storage/enigma/EnigmaSave.java#L64

Optionals

https://github.com/2menc/OOP25-C_F_U_/blame/5ee38ae834dff60ca6f23568f8a313602432
src/main/java/it/unibo/impl/templates/EnigmaTemplate.java#L111

Uso di librerie di terze parti

snakeYAML per salvataggio e caricamento di dati su file in formato yaml:
https://github.com/2menc/OOP25-C_F_U_/blob/2bf78144eb2dd249844721b7b3534d1e938f2
src/main/java/it/unibo/storage/enigma/EnigmaSave.java#L44C1-L45C1

3.2.2 Quaranta Federico

lambda

https://github.com/2menc/OOP25-C_F_U_/blob/2bf78144eb2dd249844721b7b3534d1e938f2
src/main/java/it/unibo/storage/rooms/RoomSave.java#L89

Optional

https://github.com/2menc/OOP25-C_F_U_/blob/55e2a7b69c408125177d6f403f18e057549d7
src/main/java/it/unibo/impl/RoomManagerImpl.java#L87

snakeYAML

Uso di snakeYAML per salvataggio di dati su un formato di file yaml https:
//github.com/2menc/OOP25-C_F_U_/blame/2bf78144eb2dd249844721b7b3534d1e938f2905/
src/main/java/it/unibo/storage/rooms/RoomSave.java#L85

3.2.3 Pezzolesi Luca

Optionals

Permalink: https://github.com/2menc/OOP25-C_F_U_/blame/55e2a7b69c408125177d6f403f
src/main/java/it/unibo/core/GameEngine.java#L56

Java Serialization

Permalink: https://github.com/2menc/OOP25-C_F_U_/blob/55e2a7b69c408125177d6f403f18
src/main/java/it/unibo/storage/roommanager/RoomManagerStorage.java#
L59

https://github.com/2menc/OOP25-C_F_U_/blob/55e2a7b69c408125177d6f403f18e057549d7
src/main/java/it/unibo/impl/RoomManagerImpl.java#L19

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Pezzolesi Luca

Lavorare su questo progetto è stato un momento importante per sperimentare il lavoro di squadra e la collaborazione tra più persone. Come progetto di piccole dimensioni risulta valido, tuttavia sarebbe stato opportuno dedicare maggiore attenzione alla fase di progettazione iniziale e procedere in modo più sincronizzato durante lo sviluppo. All'interno del progetto mi sono occupato della gestione dell'input, dell'implementazione dei metodi all'interno della funzione run del game engine, della realizzazione della GUI e dello sviluppo del sistema di salvataggio. Questa esperienza mi ha permesso di comprendere meglio l'importanza dell'organizzazione e della coordinazione all'interno di un team di lavoro.

4.1.2 Valente Domenico

Mi ritengo abbastanza soddisfatto del mio lavoro, in particolare mi ritengo soddisfatto del sistema di creazione di enigmi e stanze. Il mio ruolo è stato quello di creare la base su cui modellare l'ambiente di gioco. Non escludo la possibilità di continuare a lavorare al progetto in futuro, concentrandomi sulla possibilità per l'utente finale di creare nuovi ambienti di gioco, migliorando e ottimizzando il sistema già esistente.

4.1.3 Quaranta Federico

Sono complessivamente soddisfatto del lavoro svolto, che mi ha permesso di consolidare i concetti della programmazione a oggetti e di comprendere me-

glio le dinamiche di collaborazione necessarie per gestire un progetto in team. Nonostante qualche rallentamento in determinate fasi dello sviluppo, l'esperienza è stata fondamentale per migliorare sia le mie competenze tecniche che la mia capacità di gestione del tempo.

Capitolo 5

Guida utente

L'obiettivo del gioco è muoversi all'interno dell'ambiente, usando i **tasti direzionali**, risolvere gli enigmi, proseguire nelle varie stanze trovando le chiavi all'interno di alcuni enigmi per aprire la porta finale e finire il gioco

Capitolo 6

Esercitazioni di laboratorio

6.0.1 Pezzolesi Luca

- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=206731#p284100;
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=207193#p285028;
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=207921#p286148;
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=208718#p287241;
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=209589#p288508;
- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=210617#p289660;
- Laboratorio 12: https://virtuale.unibo.it/mod/forum/discuss.php?d=211539#p290698;